Lecture 13: other modules & prospects

October 31, 2014

Where are we?

```
int main() {
 #input_init(pfc,ppr,pba,pth,ppt,ptr,ppm,psp,pnl,ple,pop);
 #background_init(ppr,pba);
 #thermodynamics_init(ppr,pba,pth);
 #perturb_init(ppr,pba,pth,ppt);
 #primordial_init(ppr,ppt,ppm);
 nonlinear_init(ppr,pba,pth,ppt,ppm,pnl);
 transfer_init(ppr,pba,pth,ppt,pnl,ptr);
 spectra_init(ppr,pba,ppt,ppm,pnl,ptr,psp);
 lensing_init(ppr,ppt,psp,pnl,ple);
 #output_init(pba,pth,ppt,ppm,ptr,psp,pnl,ple,pop);
}
```

The nonlinear module

```
source/nonlinear.c
```

The nonlinear module

The goal is to

- compute factors $R^{NL}(k,\tau) = \delta_m^{NL}/\delta_m^L$,
- store them in `pnl->nl_corr_density[...]`,
- in the transfer and spectra module in charge of C_l's and $P(k)$, multiply everywhere $S_i(k,\tau)$ with $R^{NL}(k,\tau)$.

The nonlinear module

The goal is to

- compute factors $R^{NL}(k,\tau) = \delta_m^{NL}/\delta_m^{L}$,
- store them in `pnl->nl_corr_density[...]`,
- in the transfer and spectra module in charge of C_l's and $P(k)$, multiply everywhere $S_i(k,\tau)$ with $R^{NL}(k,\tau)$.

Module only features HALOFIT so far (including Takahashi et al. `arxiv:1208.2701`, extended to massive neutrinos by S. Bird in 2014, following the method of Bird et al. 2011).

```
non linear = halofit (or blank)
```

The nonlinear module

The goal is to

- compute factors $R^{NL}(k,\tau) = \delta_m^{NL}/\delta_m^L$,
- store them in `pnl->nl_corr_density[...]`,
- in the transfer and spectra module in charge of C_l's and $P(k)$, multiply everywhere $S_i(k,\tau)$ with $R^{NL}(k,\tau)$.

Module only features HALOFIT so far (including Takahashi et al. `arxiv:1208.2701`, extended to massive neutrinos by S. Bird in 2014, following the method of Bird et al. 2011).

```
non linear = halofit (or blank)
```

In project:

- other fitting formulas,
- restore one-loop PT,
- linking with recent codes of Scoccimarro et al., Bernardeau et al., Pietroni et al. featuring various renormalisation approaches.

The transfer module

`source/transfer.c`

The transfer module

The goal is to compute harmonic transfer functions by performing several integrals of the type

$$\Delta_l^X(q) = \int d\tau \;\; S_X(k(q), \tau) \;\; \phi_l^X(q, (\tau_0 - \tau))$$

for each mode, initial conditions, and several types of source functions. In flat space $k = q$.

Calculation done for few values of l (controlled by precision parameters). C_l's are interpolated later.

The transfer module

$$\Delta_l^X(q) = \int d\tau \;\; S_X(k,\tau) \;\; \phi_l^X(q,(\tau_0-\tau))$$

In flat space and for CMB transfer functions:

mode	type	source	Bessel
scalar	temperature T0	S_T^0	$j_l(x), x \equiv k(\tau_0-\tau)$
	temperature T1	S_T^1	j_l'
	temperature T2	S_T^2	$\frac{1}{2}(3j_l''+j_l)$
	polarisation E	S_P	$(...)\frac{j_l}{x^2}$
tensor	temperature T2	S_T^2	$(...)\frac{j_l''}{x^2}$
	polarisation E	S_P	$\frac{1}{4}\left(-j_l+j_l''+2\frac{j_l}{x^2}+4\frac{j_l'}{x}\right)$
	polarisation B	S_P	$\frac{1}{2}\left(j_l'+2\frac{j_l}{x}\right)$
vector	...	...	...

Much more complicated in non-flat space, and involving hyperspherical Bessel functions, different for each q.

The transfer module

$$\Delta_l^X(q) = \int d\tau \;\; S_X(k,\tau) \;\; \phi_l^X(q,(\tau_0-\tau))$$

In flat space and for LSS transfer functions:

mode	type	source	bessel
scalar	density in bin i	$W_i(\tau)\delta_m(\tau,k) + \ldots$	j_l
	CMB lensing	$-\tilde{W}_l(\tau,\tau_{\rm rec})(\phi(\tau,k)+\psi(\tau,k))$	j_l
	(with	$\tilde{W}_l = \frac{(\tau-\tau_{\rm rec})}{(\tau_0-\tau)/(\tau_0-\tau_{\rm rec})}$)	
	lensing in bin i	$-\int d\tau' W_i(\tau')\tilde{W}_l(\tau,\tau')(\phi(\tau,k)+\psi(\tau,k))$	j_l

Choose window functions $W_i(z)$ with input parameters:

```
selection=gaussian (dirac, tophat)
selection_mean = 1.,2.,3.
selection_width = 0.5
non_diagonal=1
```

Look also at options in explanatory.ini for global selection function
`dNdz_selection=analytic (filename)`, `dNdz_evolution`, `bias`.

Density source functions

In CLASS v$\geq$2.1, the galaxy number count C_l's refer to

$$\begin{aligned}\Delta(\mathbf{n}, z) = {} & D_g + \Phi + \Psi + \frac{1}{\mathcal{H}}\left[\Phi' + \partial_r(\mathbf{V}\cdot\mathbf{n})\right] \\ & + \left(\frac{\mathcal{H}'}{\mathcal{H}^2} + \frac{2}{r_S\mathcal{H}}\right)\left(\Psi + \mathbf{V}\cdot\mathbf{n} + \int_0^{r_S} dr(\Phi' + \Psi')\right) \\ & + \frac{1}{r_S}\int_0^{r_S} dr\left[2 - \frac{r_S - r}{r}\Delta_\Omega\right](\Phi + \Psi).\end{aligned}$$

(gauge-independent density, redshift space distorsions, Doppler effect, lensing, gravitational corrections). However, by default, only the first term will be computed.

These effects have been first developed in the branch CLASSgal, arXiv:1307.1459, `http:`*//cosmology.unige.ch/tools/*, and then imported into the main code.

The transfer module

Here, a single source in the perturbation module (e.g. $\delta_m(\tau, k)$) leads to several sources in the transfer module (e.g. $W_i(\tau)\delta_m(\tau, k)$). Hence, the code considers that perturbation types and transfer types are two different things, with a correspondance

```
  int index_tt_t0;            /* inferred from  index_tp_t0 */
  int index_tt_t1;            /* inferred from  index_tp_t1 */
  int index_tt_t2;            /* inferred from  index_tp_t2 */
  int index_tt_e;             /* inferred from  index_tp_p  */
  int index_tt_b;             /* inferred from  index_tp_p  */
  int index_tt_lcmb;          /* from index_tp_phi_plus_psi */
  int index_tt_density;       /* from index_tp_delta_m, ... */
  ....
  int index_tt_lensing;       /* from index_tp_phi_plus_psi */
  ....

  int * tt_size;
```

The function transfer_init()

- define dynamical indices with `transfer_indices_of_transfers()`.
- interpolate all sources along k with `transfer_perturbation_source_spline()`, k grid is finer in transfer module than in perturbation module.
- compute all flat Bessel functions
- loop over q (or k in flat space).
- : if non-flat, compute hyperspherical Bessel functions for this q
- : loop over modes, initial conditions, types
- : : translate `_tp_` in `_tt_` source, and eventually redefine time sampling
- : : loop over l
- : : : integrate transfer function, or approximate it by Limber, or by zero

$$\Delta_l^X(q) = \int d\tau \;\; S_X(k,\tau) \;\; \phi_l^X(q,(\tau_0-\tau))$$

Plotting a transfer function with test/test_transfer.c

```c
input_init_from_arguments(argc, argv,&pr,&ba,&th,&pt,&tr,&pm
    ,&sp,&nl,&le,&op,errmsg);

background_init(&pr,&ba);
thermodynamics_init(&pr,&ba,&th);
perturb_init(&pr,&ba,&th,&pt);
transfer_init(&pr,&ba,&th,&pt,&tr);

/* choose a mode (scalar, tensor, ...) */
int index_mode=pt.index_md_scalars;

/* choose a type (temperature, polarization, grav. pot.,
    ...) */
int index_type=pt.index_tt_t0;

/* choose an initial condition (ad, bi, cdi, nid, niv, ...)
    */
int index_ic=pt.index_ic_ad;
```

Plotting a transfer function with test/test_transfer.c

```c
output=fopen("output/test.trsf","w");

for (index_l=0; index_l<tr.l_size[index_mode]; index_l++) {
   for (index_q=0; index_q<tr.q_size; index_q++) {

      transfer = tr.transfer[index_mode]
          [((index_ic * tr.tt_size[index_mode] + index_type)
           * tr.l_size[index_mode] + index_l)
           * tr.q_size + index_q];

      fprintf(output,"%d %e %e %e\n",
              tr.l[index_l],
              tr.q[index_q],
              tr.k[index_mode][index_q],
              transfer);
   }
}
```

For instance you can type:

```
> make test_transfer
> ./test_transfer my_input.ini
```

The spectra module

source/spectra.c

The function spectra_init()

The function `spectra_init()` calls (at most) three functions:

- `spectra_pk` computes the linear and non-linear matter power spectrum

  $$P_L(k,z) = (\delta_m(k,\tau(z)))^2\,\mathcal{P}(k)$$

  or in the case of several initial conditions,

  $$P_L(k,z) = \sum_{ij} \delta_m^i(k,\tau(z))\,\delta_m^j(k,\tau(z))\mathcal{P}_{ij}(k)$$

  (same for $P_{NL}(k,z)$ with extra factor R_{NL}^2). Result stored in `psp->ln_pk` and `psp->ln_pk_nl`.
- `spectra_sigma()` computes mean variance in sphere of radius R. By default, code calls it to get $\sigma_8(z=0)$ and stores it in `psp->sigma8`

The function spectra_init()

- `spectra_cl` computes the harmonic power spectra (formulae below holds in flat space)

$$C_l^{XY} = 4\pi \sum_{ij} \int \frac{dk}{k} \Delta_l^X(k) \Delta_l^Y(k) \mathcal{P}(k)$$

or in the case of several initial conditions,

$$C_l^{XY} = 4\pi \sum_{ij} \int \frac{dk}{k} \frac{1}{2} \left[\Delta_l^{iX}(k) \Delta_l^{jY}(k) + \Delta_l^{iX}(k) \Delta_l^{iY}(k) \right] \mathcal{P}_{ij}(k)$$

for $XY \in \{TT, TE, EE, BB, PP, TP, EP, N_i N_j, TN_i, PN_i, L_i L_j, TL_i, N_i L_j\}$ where $P \equiv$ CMB lensing, $N_i \equiv$ galaxy number count in i-th bin, and $L_i \equiv$ galaxy lensing in i-th bin.
Calculation takes place only for few values of l, later result can be interpolated at any l.

The result is stored in `psp->cl`.

External functions in spectra

Functions called later by `output` module, but you might need to use them directly when embedding CLASS in your own code or when writing your own `main` function.

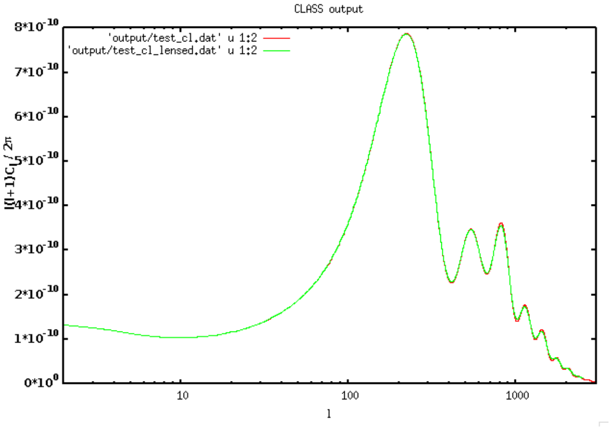
- `spectra_cl_at_l()` returns C_l's (either linear or non-linear) at a given l. Result can be a big array because returns result for each type, each i.c., each mode, and the total for each type.
- `spectra_pk_at_z()` returns $P_L(k,z)$ at given z for all k's.
- `spectra_pk_at_k_and_z()` returns $P_L(k,z)$ at given z and k.
- `spectra_pk_nl_at_z()` returns $P_{NL}(k,z)$ at given z for all k's.
- `spectra_pk_nl_at_k_and_z()` returns $P_{NL}(k,z)$ at given z and k.
- `spectra_sigma()` returns σ_R, i.e. the variance of matter fluctuations in a sphere of radius R, like the usual σ_8.
- `spectra_bandpower(l_1,l_2)` returns the bandpower $\sum_{l_1<l<l_2}(2l+1)C_l$

The lensing module

`source/lensing.c`

The function lensing_init()

Given the unlensed CMB spectra $C_l^{TT,TE,EE,BB}$ and the spectrum of the lensing potential C_l^{PP}, the goal is to compute the lensed spectra $\tilde{C}_l^{TT,TE,EE,BB}$.



The function lensing_init()

- follows the all-sky method of A. Challinor and A. Lewis.
- implemented in CLASS by S. Prunet.
- differs from CAMB only through numerical implementation (quadrature weights, etc.), not methodology. Results agree very well.
- module is switched on/off with `lensing = yes, no`. But it also requires at least `output= tCl, lCl` or `output= pCl, lCl` or `output= tCl, pCl, lCl` to output lensed spectra.

At the end of `lensing_init()`, `ple->cl_lens` contains a replica of tables in `psp->cl`, excepted that $C_l^{TT,TE,EE,BB}$ are replaced by their lensed counterpart.

External function: `lensing_cl_at_l()`, works like `spectra_cl_at_l()`, but returns only total lensed spectra (individual lensed spectra make no sense).

Called by `output` module or by external code. Contains really observable quantities: called by `classy.pyx` and Monte Python.